

PATENT ABSTRACTS OF JAPAN

(11)Publication number : 2001-273150
 (43)Date of publication of application : 05.10.2001

i1)Int.Cl.

G06F 9/45

1)Application number : 2000-087626
 2)Date of filing : 27.03.2000

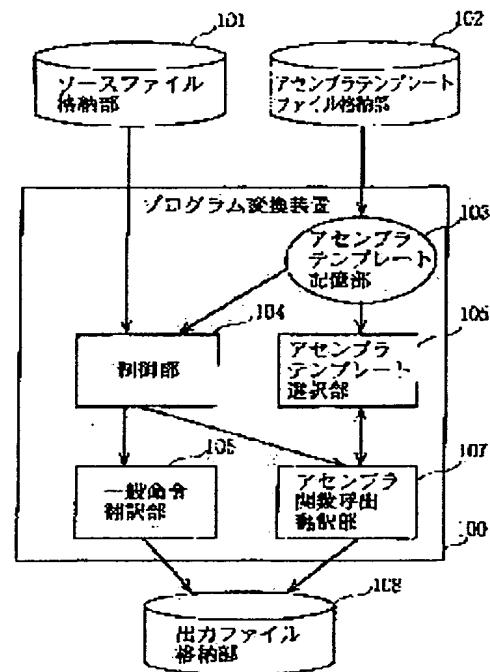
(71)Applicant : MATSUSHITA ELECTRIC IND CO LTD
 (72)Inventor : SAKATA TOSHIYUKI
 URUSHIBARA SEIICHI
 HASHIGUCHI WATARU

i4) DEVICE FOR CONVERTING PROGRAM

i7)Abstract:

PROBLEM TO BE SOLVED: To provide a program conversion device that converts access description into an optimum assembler code when the access description of an assembler template function being a description format to be developed into an assembler code is included in a source program.

SOLUTION: When a controlling part 104 detects the access description of the assembler template function, an assembler template selecting part 105 obtains a plurality of definitions corresponding to the assembler template function accessed by the access description from an assembler template storing part 103, decides whether or not the respective definitions are usable according to whether an argument in the access description is a variable or a constant, the number of performance cycles of an assembler code specified by a definition is calculated with respect to each of usable definition, and one definition in which the number of performance cycles is the smallest is selected. An assembler function access translating part 107 generates an assembler code on the selected definition and outputs the assembler code to an output file storing part 108.



LEGAL STATUS

Date of request for examination]

Date of sending the examiner's decision of rejection]

Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

Date of final disposal for application]

Patent number]

Date of registration]

Number of appeal against examiner's decision of rejection]

Date of requesting appeal against examiner's decision of rejection]

Date of extinction of right]

(19)日本国特許庁 (JP)

(12) 公開特許公報 (A)

(11)特許出願公開番号

特開2001-273150

(P2001-273150A)

(43)公開日 平成13年10月5日 (2001.10.5)

(51)Int.Cl.⁷

G 0 6 F 9/45

識別記号

F I

C 0 6 F 9/44

マークト[®](参考)

3 2 2 E 5 B 0 8 1

審査請求 未請求 請求項の数10 O L (全 14 頁)

(21)出願番号 特願2000-87626(P2000-87626)

(22)出願日 平成12年3月27日 (2000.3.27)

(71)出願人 000005821

松下電器産業株式会社

大阪府門真市大字門真1006番地

(72)発明者 坂田 俊幸

大阪府門真市大字門真1006番地 松下電器
産業株式会社内

(72)発明者 漆原 誠一

大阪府門真市大字門真1006番地 松下電器
産業株式会社内

(74)代理人 100090446

弁理士 中島 司朗 (外1名)

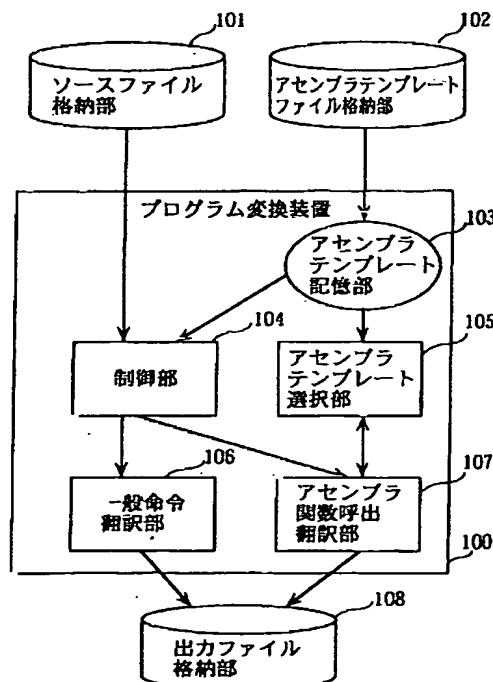
最終頁に続く

(54)【発明の名称】 プログラム変換装置

(57)【要約】

【課題】 ソースプログラム中にアセンブラーコードに展開される記述形式であるアセンブラテンプレート関数の呼出記述が含まれている場合に、呼出記述を最適なアセンブラーコードに変換するプログラム変換装置を提供する。

【解決手段】 制御部104がアセンブラテンプレート関数の呼出記述を検出したとき、アセンブラテンプレート選択部105は、呼出記述で呼び出されるアセンブラテンプレート関数に対応する複数の定義をアセンブラテンプレート記憶部103から得て、呼出記述中における引数が変数か定数かの別により各定義が使用可能か否かを判断し、使用可能な各定義に対して定義で特定されるアセンブラーコードの実行サイクル数を求め、実行サイクル数の最も小さい1つの定義を選択する。アセンブラー関数呼出翻訳部107は選択された定義に基づいてアセンブラーコードを生成して出力ファイル格納部108に出力する。



【特許請求の範囲】

【請求項1】 高級言語で記述されたソースプログラムをアセンブラーコードに変換するプログラム変換装置であって、アセンブラテンプレート関数についての複数の定義を記憶するテンプレート記憶手段と、前記ソースプログラムにおけるアセンブラテンプレート関数の呼出記述を検出する呼出検出手段と、前記呼出検出手段により前記呼出記述が検出された場合に、当該呼出記述により呼び出されるアセンブラテンプレート関数についての複数の定義のうち、最適な1つを選択する選択手段と、前記呼出検出手段により検出された前記呼出記述を、前記選択手段により選択されたアセンブラテンプレート関数の定義に基づいてアセンブラーコードに変換する変換手段とを備えることを特徴とするプログラム変換装置。

【請求項2】 前記呼出検出手段は、前記ソースプログラム中の関数の呼出記述を検出して、当該呼出記述により呼び出される関数についての定義が前記テンプレート記憶手段に記憶されているか否かを判断し、定義が記憶されている場合に当該呼出記述をアセンブラテンプレート関数の呼出記述として検出することを特徴とする請求項1記載のプログラム変換装置。

【請求項3】 前記テンプレート記憶手段は、前記アセンブラテンプレート関数の定義毎に、関数への引数についての条件がある場合にはその条件を示す条件情報をも記憶しており、前記選択手段は、前記条件情報に従い、前記呼出記述に含まれる関数への引数の記述に応じて前記選択を行うことを特徴とする請求項1又は2記載のプログラム変換装置。

【請求項4】 前記条件情報は、引数が即値でなければならない旨を示す情報であり、

前記選択手段は、前記条件情報に従い、前記呼出記述に含まれる関数への引数の記述が即値であるか否かに応じて前記選択を行うことを特徴とする請求項3記載のプログラム変換装置。

【請求項5】 前記条件情報は、引数が所定範囲値内の即値でなければならない旨を示す情報であり、

前記選択手段は、前記条件情報に従い、前記呼出記述に含まれる関数への引数の記述が所定範囲内の即値であるか否かに応じて前記選択を行うことを特徴とする請求項4記載のプログラム変換装置。

【請求項6】 前記テンプレート記憶手段は、前記アセンブラテンプレート関数の定義毎に、最適なものを選択するための基準となる評価値をも記憶しており、

前記選択手段は、前記各定義についての評価値に基づいて、前記選択を行うことを特徴とする請求項1又は2記載のプログラム変換装置。

【請求項7】 前記テンプレート記憶手段は、前記定義

毎に当該定義が特定するアセンブラーコードについての実行サイクル数を示す情報を前記評価値として記憶しており、

前記選択手段は、前記複数の定義のうち、実行サイクル数が最小であるアセンブラーコードに変換されることになる1つの定義を選択することを特徴とする請求項6記載のプログラム変換装置。

【請求項8】 前記テンプレート記憶手段は、前記定義毎に当該定義が特定するアセンブラーコードについてのコードサイズを示す情報を前記評価値として記憶しており、

前記選択手段は、前記複数の定義のうち、コードサイズが最小であるアセンブラーコードに変換されることになる1つの定義を選択することを特徴とする請求項6記載のプログラム変換装置。

【請求項9】 前記選択手段は、前記呼出記述を仮に前記定義それぞれにより特定されるアセンブラーコードに変換する場合に、引数をレジスタに設定するためのコード並びにレジスタ退避及び復元のためのコードのうち必要となるコードがあればそのコードをも含めて得られる変換結果のコードについての実行サイクル数又はコードサイズに基づいて判断することにより、前記選択を行うことを特徴とする請求項1又は2記載のプログラム変換装置。

【請求項10】 アセンブラテンプレート関数についての複数の定義を記憶するメモリを備えるコンピュータに、高級言語で記述されたソースプログラムをアセンブラーコードに変換するプログラム変換処理を実行させるための制御プログラムを記録した記録媒体であって、前記プログラム変換処理は、

前記ソースプログラムにおけるアセンブラテンプレート関数の呼出記述を検出する呼出検出ステップと、前記呼出検出ステップにより前記呼出記述が検出された場合に、前記メモリに記憶されており当該呼出記述により呼び出されるアセンブラテンプレート関数についての複数の定義のうち、最適な1つを選択する選択ステップと、

前記呼出検出ステップにより検出された前記呼出記述を、前記選択ステップにより選択されたアセンブラテンプレート関数の定義に基づいてアセンブラーコードに変換する変換ステップとを含むことを特徴とする記録媒体。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】 本発明は、高級言語で記述されたソースプログラムを特定のプロセッサ用のアセンブラーコードに変換するプログラム変換装置に関し、特に、別途定義してあるアセンブラーコードに変換されるべき関数呼出記述を含むソースプログラムを、変換対象とするプログラム変換装置に関する。

【0002】

【従来の技術】一般にプログラム開発者等によるプログラムの作成は、C言語、C++言語等の高級言語を用いてなされ、高級言語を用いて作成されたプログラムは、コンパイラ等のプログラム変換装置によってアセンブラーコードに変換され最終的にはプロセッサが実行可能な機械語命令列に変換され利用される。

【0003】ところで近年、画像処理等のマルチメディア処理を高速に行なうために積和演算、飽和演算等のデジタル信号処理用の命令（以下、「DSP命令」という。）をサポートするマイクロプロセッサが製造されるようになり、このようなマイクロプロセッサが情報家電機器やパーソナルコンピュータに搭載されている。しかしながら、C言語、C++言語等の高級言語はDSP命令を想定しておらずDSP命令に対応する演算子を備えていない。

【0004】従って、DSP命令を使用する処理を実行するプログラムを作成するためには、DSP命令を使用する処理の部分をアセンブラー言語で記述しなければならない。従来のプログラム変換装置には、高級言語のソースプログラム中においてアセンブラーコードにより実現される処理を記述するために、アセンブランプレート関数に対する呼出記述を用いることができるようとしたものがある。例えばGAIOTEKノロジ社のコンパイラである。アセンブランプレート関数とは、アセンブラーコードを決定するために用いられる記述であり、形式的に関数の形式をとるものである。このアセンブランプレート関数の内容を、即ちアセンブラーコードを決定するための情報等を、定義したものをアセンブランプレートという。つまりアセンブランプレートは、アセンブランプレート関数を定義するための記述、即ちアセンブランプレート関数の内容を特定するための記述である。

【0005】このようなプログラム変換装置は、ソースプログラム中にアセンブランプレート関数の呼出記述が記述されている場合には、その呼出記述における引数及び戻り値に即して、アセンブランプレート関数の内容となるアセンブラーコードを決定し、この決定したアセンブラーコードをそのアセンブランプレート関数の呼出記述の部分の翻訳結果とする。即ち、このプログラム変換装置により、アセンブランプレート関数の呼出記述は、サブルーチンを呼び出す命令を含むアセンブラーコードに翻訳されるのではなく、その位置に直接置かれるアセンブラーコードに翻訳される。

【0006】図10は、従来のアセンブランプレートの内容例を示す図である。同図に示すようにアセンブランプレートの内容は、アセンブランプレート関数の定義であり、各行の意味は以下の通りである。「asm _template」はアセンブランプレートであることを表す。「fir」は、アセンブランプレート関数の関数名を表す。なお、1つの関数名で特定されるア

センブランプレート関数の定義記述は1つのみである。即ち、同一関数名で2以上のアセンブランプレート関数の定義を行うことはできない。

【0007】「\$p1 = {p0, p1, p2, p3}」は、「\$p1」という仮想レジスタがアセンブランプレート関数の第1引数を表し、「{p0, p1, p2, p3}」が第1引数の割り当て候補のレジスタを表す。ここで、仮想レジスタとは、レジスタに変換される変数をいう。プログラム変換装置は、ソースプログラムの翻訳に際して、アセンブランプレート関数の呼出記述における第1引数に対応するものとして、割り当て候補のレジスタうちのいずれか1つのレジスタを決定して割り当てることになり、結果的に第1引数の示す値はその決定した1つのレジスタに格納されることになる。

【0008】「\$p2 = {p4, p5, p6, p7}」は、「\$p2」という仮想レジスタがアセンブランプレート関数の第2引数を表し、「{p4, p5, p6, p7}」が第2引数の割り当て候補のレジスタを表す。「\$p3 = {r0, r1, r2, r3}」は、「\$p3」という仮想レジスタがアセンブランプレート関数の第3引数を表し、「{r0, r1, r2, r3}」が第3引数の割り当て候補のレジスタを表す。

【0009】「\$t1 = {a0, a1}」は、「\$t1」という仮想レジスタがアセンブランプレート関数内で使用される1番目の一時資源を表し、「{a0, a1}」が一時資源への割り当て候補のレジスタを表す。なお一時資源は、一時的に使用するレジスタを意味する。「ret = {\$t1}」は、アセンブランプレート関数内で使用される1番目の一時資源が関数の戻り値とされることを表す。

【0010】「kill = {\$p1, \$p2}」は、第1引数、第2引数の値が関数内で変更されることを表す。「code = {...}」は、アセンブランプレート関数の定義の本体的な記述であり、引数及び一時資源を用いて表されたアセンブラーコードである。「repn \$p3」は第3引数の回数だけ次の命令を繰り返すという命令であり、「mac \$t1, m(\$p1, 1), m(\$p2, 1)」は第1引数であるアドレス値の指し示すメモリの内容と第2引数であるアドレス値の指し示すメモリの内容との積と一時資源との和を一時資源に格納した後に、第1引数であるアドレス値及び第2引数であるアドレス値をそれぞれ増加する命令である。

【0011】即ち、この「fir」という関数名のアセンブランプレート関数は、第1引数であるアドレスから順にメモリに格納されている数値列と、第2引数であるアドレスから順にメモリに格納されている数値列との内積を第3引数で示される要素数分だけ計算した結果の総和を戻り値として返却する関数である。図6は、アセンブランプレート関数の呼出記述を含むソースプログラムの例を示す図である。

【0012】同図に示すソースプログラムは、C言語で記述されたものであり、「`fir (v1, v2, n)`」及び「`fir (v3, v4, 10)`」が上述した「`fir`」という関数名のアセンブリテンプレート関数の呼出記述である。以下、図9を用いて、従来のプログラム変換装置900について説明する。図9は、従来のプログラム変換装置900の機能ブロック図である。

【0013】なお、同図中にはプログラム変換装置の入力ファイルの格納先であるソースファイル格納部901及びアセンブリテンプレートファイル格納部902と、出力ファイルの格納先である出力ファイル格納部907をも示している。プログラム変換装置900は、ソースプログラムをアセンブリコードに変換するもので、ハードウェア的にはCPU、メモリ、ハードディスク装置等を備えるパーソナルコンピュータで構成され、機能的にはアセンブリテンプレート記憶部903と、制御部904と、アセンブリ関数呼出翻訳部905と、一般命令翻訳部906とを備える。制御部904、アセンブリ関数呼出翻訳部905及び一般命令翻訳部906は、メモリに格納された制御プログラムがCPUにより実行されることによりその機能が実現されるものである。

【0014】ここで、ソースファイル格納部901は、アセンブリテンプレート関数の呼出記述を含むソースプログラムを格納している記憶装置である。アセンブリテンプレートファイル格納部902は、ソースプログラムから呼び出されるアセンブリテンプレート関数の定義であるアセンブリテンプレートが記述されたアセンブリテンプレートファイルを格納している記憶装置である。

【0015】アセンブリテンプレート記憶部903は、アセンブリテンプレートファイル格納部902に格納されたアセンブリテンプレートファイルからアセンブリテンプレートを読み出して記憶するメモリ領域である。制御部904は、ソースファイル格納部901からソースプログラムを読み出し、ソースプログラムの各文に逐次着目し、着目した文が、関数の呼出記述を意味する文でありかつその関数の名前がアセンブリテンプレート記憶部903に記憶されていた場合には、アセンブリ関数呼出翻訳部905を起動し、その他の場合には、一般命令翻訳部906を起動する。ここで起動とは、起動対象に機能処理の実行を開始させることをいう。

【0016】アセンブリ関数呼出翻訳部905は、制御部904により起動され、関数呼出記述で呼び出されるアセンブリテンプレート関数に対応する定義をアセンブリテンプレート記憶部903から得て、その本体的な記述である「`c o d e = { . . . }`」において使用されている「`$p1`」、「`$p2`」、「`$p3`」、「`$t1`」に対して、割り当て候補のレジスタのうちのいずれかを割り当てて引数及び戻り値の設定のコードとレジスタの退避及び復元のためのコードとを付加することによりアセンブリコードを生成して、生成したアセンブリコード

を出力ファイル907に出力する。

【0017】一般命令翻訳部906は、制御部904により起動され、一般命令翻訳処理を行なう。即ち、一般命令翻訳部906は、アセンブリテンプレート関数の呼出記述を含む文以外の文を、文の内容に応じてアセンブリコードに変換して出力ファイル格納部907に出力する。なお、プログラム変換装置900は、ソースファイル格納部901及びアセンブリテンプレートファイル格納部902からソースプログラム及びアセンブリテンプレートを読み出した後は、これらを処理効率のよい所定の内部形式に変換し、これらをアセンブリコードへ翻訳する処理を実行する。

【0018】図11は、図6に示したソースプログラムを従来のプログラム変換装置900により変換した結果として得られるアセンブリコードを示す図である。なお、アセンブリテンプレートファイル格納部902に、図10に示したアセンブリテンプレートが格納されていた場合に、図11に示すアセンブリコードが得られることになる。

【0019】図6に示したソースプログラムにおいて、アセンブリテンプレート関数の1回目の使用については全ての引数を変数とし、2回目の使用については第1引数と第2引数とを変数とし第3引数を定数「10」としている。しかし、このソースプログラムの変換結果として生成されたアセンブリコードは、アセンブリテンプレート関数の引数に変数を用いられたか定数を用いられたかに関係なく、図11に示すように1回目の使用に相当する部分も2回目の使用に相当する部分も形式的には同じ命令列から構成されるものとなる。

【0020】このように、従来のプログラム変換装置は、同一のアセンブリテンプレート関数を使用する部分についてはいずれも形式的に同じ命令列から構成されるアセンブリコードに変換する。

【0021】

【発明が解決しようとする課題】上述した従来のプログラム変換装置は、ソースプログラムを最適なアセンブリコードに変換する、即ち最適なアセンブリコードを生成するという観点からは、以下に示す問題がある。プロセッサが実行可能な命令セットには、処理対象がレジスタの内容であるか定数であるかによって異なる命令が用意されている場合があり、効果的に各命令を使い分けることにより最適なアセンブリコードの生成は可能になる。このような異なる命令が用意されている場合において、例えば定数を処理対象とする命令が利用できるにも関わらず、定数をレジスタに格納した後にレジスタを処理対象とする命令を利用したならば、最適なアセンブリコードの生成を行ったとはいえない。

【0022】従来のプログラム変換装置によれば、アセンブリテンプレート関数の呼出記述は、どのような引数を用いた記述であっても、形式的に同じ命令列から構成

されるアセンブラコードに変換されてしまい、引数が変数であるか定数であるかに応じて最適なアセンブラコードに変換されるのではない。また、機能的に同様な処理を行なうもので関数名が異なる複数のアセンブラテンプレート関数を用意しておき、引数が変数であるか定数であるかといった状況に応じて、ソースプログラムの作成者がどのアセンブラテンプレート関数を使用するかを選択して関数の呼出記述を行なう方法が考えられるが、作成者に選択の負担を課すため生産性の低下を招く上、作成者が必ずしも最適な選択を行うとは限らないという問題がある。

【0023】そこで、本発明は、かかる問題点に鑑みてなされたものであり、アセンブラテンプレート関数の呼出記述における引数が変数であるか定数であるか等の呼出記述の内容に応じてアセンブラテンプレート関数の呼出記述を最適なアセンブラコードに変換するプログラム変換装置を提供することを目的とする。更に、本発明は、アセンブラテンプレート関数の呼出記述の内容に応じて最適化のために命令を使い分ける他に、コードサイズの面や実行サイクル数の面のうちどの面について最適化を図るかという設定に応じて命令を使い分けることにより、アセンブラテンプレート関数の呼出記述を最適なアセンブラコードに変換するプログラム変換装置を提供することをも目的とする。

【0024】

【課題を解決するための手段】上記目的を達成するため、本発明に係るプログラム変換装置は、高級言語で記述されたソースプログラムをアセンブラコードに変換するプログラム変換装置であって、アセンブラテンプレート関数についての複数の定義を記憶するテンプレート記憶手段と、前記ソースプログラムにおけるアセンブラテンプレート関数の呼出記述を検出する呼出検出手段と、前記呼出検出手段により前記呼出記述が検出された場合に、当該呼出記述により呼び出されるアセンブラテンプレート関数についての複数の定義のうち、最適な1つを選択する選択手段と、前記呼出検出手段により検出された前記呼出記述を、前記選択手段により選択されたアセンブラテンプレート関数の定義に基づいてアセンブラコードに変換する変換手段とを備えることを特徴とする。

【0025】ここで、アセンブラテンプレート関数の定義とは、アセンブラコードを特定する内容の情報をいう。これにより、アセンブラテンプレート関数についての複数の定義のうちの1つが選択されるので、このプログラム変換装置を利用する場合に、1つのアセンブラテンプレート関数に対して複数の定義を用意しておけば、ソースプログラム中のアセンブラテンプレート関数の呼出記述を、その記述内容等に応じて最適なアセンブラコードに変換することや、コードサイズ又は実行サイクル数の面から最適なアセンブラコードに変換することが可能になる。例えば、ソースプログラム中のアセンブラテ

ンプレート関数の呼出記述における引数が変数か定数か等に応じ、又は、変換をコードサイズ優先で行うか実行サイクル数優先で行うか等の設定に応じて、アセンブラテンプレート関数の呼出記述を、コードサイズの小さい又は実行サイクル数の少ないといった最適なアセンブラコードに変換することが可能になる。

【0026】

【発明の実施の形態】以下、本発明の実地の形態に係るプログラム変換装置について、図面を用いて説明する。<構成>図1は、本発明の実施の形態に係るプログラム変換装置100の機能ブロック図である。

【0027】なお、図1にはソースファイル格納部101、アセンブラテンプレートファイル格納部102、出力ファイル格納部108も併せて記載している。ここで、ソースファイル格納部101は、アセンブラテンプレート関数の呼出記述を含むC言語のソースプログラムを格納している記憶装置である。アセンブラテンプレートファイル格納部102は、アセンブラテンプレート関数の定義であるアセンブラテンプレートを複数記述したアセンブラテンプレートファイルを格納している記憶装置である。

【0028】また、出力ファイル格納部108は、プログラム変換装置100により、ソースファイル格納部101に格納されたソースプログラムの変換結果として生成されるアセンブラコードを格納するための記憶装置である。プログラム変換装置100は、CPU、メモリ、ハードディスク装置等を備えるパーソナルコンピュータで構成され、メモリに格納された制御プログラムがCPUにより実行されることによりソースプログラムをアセンブラコードに変換する機能を実現するいわゆるコンパイラであり、機能的には、アセンブラテンプレート記憶部103、制御部104、アセンブラテンプレート選択部105、一般命令翻訳部106、アセンブラ関数呼出翻訳部107から構成される。なお、ソースプログラムをアセンブラコードに変換するとは、ソースプログラムに基づいて、そのソースプログラムに記述された処理内容をプロセッサに実行させるためのアセンブラコードを生成することをいう。

【0029】アセンブラテンプレート記憶部103は、アセンブラテンプレートファイル格納部102からアセンブラテンプレートを読み出して記憶するメモリ領域であり、従来のプログラム変換装置900におけるアセンブラテンプレート記憶部903と同等である。制御部104は、ソースファイル格納部101からソースプログラムを読み出し、ソースプログラムの各文に逐次着目し、着目した文が、関数の呼出記述を意味する文でありかつその関数の名前がアセンブラテンプレート記憶部103に記憶されていた場合には、アセンブラ関数呼出翻訳部107を起動し、その他の場合には、一般命令翻訳部106を起動する。なお、制御部104は、従来のP

ログラム変換装置900における制御部904と同等である。

【0030】アセンブラー関数呼出翻訳部905は、制御部904により起動され、アセンブランプレート選択部105を起動して、関数の呼出記述で呼び出されるアセンブランプレート関数に対応する定義をアセンブランプレート選択部105から受け取り、アセンブランプレート関数の定義の本体的な記述において使用されている仮想レジスタに対して、割り当て候補のレジスタのうちのいずれかを割り当てて引数及び戻り値の設定のコードとレジスタの退避及び復元のためのコードとを附加することによりアセンブラコードを生成して、生成したアセンブラコードを出力ファイル格納部108に出力する。

【0031】アセンブランプレート選択部105は、アセンブラ関数呼出翻訳部107により起動され、関数呼出記述で呼び出されるアセンブランプレート関数に対応する複数の定義、即ち複数のアセンブランプレートを、アセンブランプレート記憶部103から獲得し、それぞれのアセンブランプレートのコストを計算し、最小のコストのアセンブランプレートを選択し、アセンブラ関数呼出翻訳部107に渡す。なお、コストとは、アセンブランプレートの選択に用いるための指標値をいい、コストが低いことは、アセンブランプレート関数の翻訳結果であるアセンブラコードのコードサイズが小さい又はそのアセンブラコードの実行サイクル数が短いものであることを表す。

【0032】一般命令翻訳部106は、制御部104により起動され、アセンブランプレート関数の呼出記述を含む文以外の文を、文の内容に応じてアセンブラコードに変換して出力ファイル格納部108に出力する。なお、一般命令翻訳部106は、従来のプログラム変換装置900における一般命令翻訳部906と同等である。

【0033】なお、プログラム変換装置100は、ソースファイル格納部101及びアセンブランプレートファイル格納部102からソースプログラム及びアセンブランプレートを読み出した後は、一般のコンバイラがソースプログラムを内部形式に変換して翻訳処理するのと同様に、ソースプログラム及びアセンブランプレートを処理効率のよい所定の内部形式に変換しておき、これらをアセンブラコードへ翻訳する処理を実行する。
<データ内容>以下、アセンブランプレートファイル格納部102に格納されている複数のアセンブランプレートについて説明する。

【0034】図7は、複数のアセンブランプレートの内容例を示す図である。同図の例では、同一の関数名の2つのアセンブランプレート関数の定義記述である2つのアセンブランプレートを示している。まず、同図中のテンプレート1、即ち1つ目のアセンブランプレート関数の定義について説明する。

【0035】「`f i r`」は、アセンブランプレート関数の関数名を表す。「`$ p 1 = { p 0, p 1, p 2, p 3 }`」は、「`$ p 1`」という仮想レジスタがアセンブランプレート関数の第1引数を表し、「`{ p 0, p 1, p 2, p 3 }`」が第1引数の割り当て候補のレジスタを表す。「`$ p 2 = { p 4, p 5, p 6, p 7 }`」は、「`$ p 2`」という仮想レジスタがアセンブランプレート関数の第2引数を表し、「`{ p 4, p 5, p 6, p 7 }`」が第2引数の割り当て候補のレジスタを表す。

【0036】「`$ p 3 = { r 0, r 1, r 2, r 3 }`」は、「`$ p 3`」という仮想レジスタがアセンブランプレート関数の第3引数を表し、「`{ r 0, r 1, r 2, r 3 }`」が第3引数の割り当て候補のレジスタを表す。「`$ t 1 = { a 0, a 1 }`」は、「`$ t 1`」という仮想レジスタがアセンブランプレート関数内で使用される1番目の一時資源を表し、「`{ a 0, a 1 }`」が一時資源への割り当て候補のレジスタを表す。

【0037】「`r e t = { $ t 1 }`」は、アセンブランプレート関数内で使用される1番目の一時資源が関数の戻り値とされることを表す。「`k i 1 1 = { $ p 1, $ p 2 }`」は、第1引数、第2引数の値が関数内で変更されることを表す。「`c o d e = { . . . }`」の内容として記述された「`r e p n $ p 3`」及び「`m a c $ t 1, m ($ p 1, 1), m ($ p 2, 1)`」は、アセンブランプレート関数の定義の本体的な記述であり、「`$ p 1`」、「`$ p 2`」、「`$ p 3`」、「`$ t 1`」という仮想レジスタを用いて表されたアセンブラコードである。

【0038】「`c o s t = 2`」は、コストが2であることを表す。次に、同図中のテンプレート2、即ち関数名「`f i r`」の2つ目のアセンブランプレート関数の定義について、テンプレート1と異なる点についてのみ説明する。「`$ p 3 = { i m m }`」は、「`$ p 3`」という仮想レジスタがアセンブランプレート関数の第3引数を表し、「`{ i m m }`」が第3引数の割り当て候補は定数値であることを表す。

【0039】「`c o d e = { . . . }`」の内容として記述された「`m a c r $ s p 3, $ t 1, m ($ p 1, 1), m ($ p 2, 1)`」は、アセンブランプレート関数の定義の本体的な記述であり、「`$ p 1`」、「`$ p 2`」、「`$ p 3`」、「`$ t 1`」という仮想レジスタを用いて表されたアセンブラコードである。「`c o s t = 1`」は、コストが1であることを表す。

<動作>以下、上述の構成を備えるプログラム変換装置100の動作について、図6に示したC言語で記述されたソースプログラムの変換を例として、説明する。

【0040】前提として、この変換対象となる図6に示したソースプログラムがソースファイル格納部101に格納されているものとし、図7に示した2つのアセンブランプレート関数の定義、即ち2つのアセンブラン

プレートが、アセンブランプレートファイル格納部102に格納されているものとする。図2は、プログラム変換装置100の動作を示すフローチャートである。

【0041】まず、プログラム変換装置100は、アセンブランプレートファイル格納部102に格納されている全てのアセンブランプレートを読み出してプログラム変換装置の内部形式に変換して、アセンブランプレート記憶部103に格納する（ステップS201）。アセンブランプレートを読み出した後に、プログラム変換装置100は、ソースファイル格納部101に格納されているソースプログラムを読み出してメモリに格納する（ステップS202）。

【0042】ソースプログラムがメモリに格納された後に、制御部104は、ソースプログラム中の各文に逐次着目して、その内容に応じて翻訳処理を行う（ステップS203～S208）。即ち、制御部104は、ソースプログラムの1文に着目し（ステップS203）、この文が関数の呼出記述であるかを判断し（ステップS204）、関数の呼出記述である場合には次にアセンブランプレート記憶部103にその呼び出す関数名が記憶されているかを判断し（ステップS205）、記憶されているときには、アセンブラン関数呼出翻訳部107を起動して、アセンブラン関数呼出翻訳処理を実行させる（ステップS206）。なお、アセンブラン関数呼出翻訳処理については、後に詳細に説明する。

【0043】図6に示すソースプログラム中の1つの文である「`resul t1=fir(v1, v2, n)`」は関数の呼出記述であり、かつ関数名「`fir`」はアセンブランプレート記憶部103に格納されているので（図7参照）、アセンブラン関数呼出翻訳処理が行われることになる。また、制御部104は、着目した文が（ステップS203）、関数の呼出記述ではない場合か（ステップS204）、又は呼び出される関数の名前がアセンブランプレート記憶部103に記憶されていない場合には（ステップS205）、一般命令翻訳部106を起動して、文の内容に応じてアセンブランコードに変換する一般命令翻訳処理を実行させる（ステップS207）。一般命令翻訳処理については、一般的コンパイラにおける翻訳処理と同様であるので、詳細な動作説明は省略する。

【0044】アセンブラン関数呼出翻訳処理（ステップS206）又は一般命令翻訳処理（ステップS207）が終了した後に、制御部104は、ソースプログラム中に未だ着目していない文があるか否かを判断し（ステップS208）、未だ着目していない文があるならば次の文に着目すべくステップS203の処理に戻る。図6に示すソースプログラム中にはまだ次の文「`resul t2=fir(v3, v4, 10)`」があるのでその文に着目することになる（ステップS203）。この文も関数呼び出しで（ステップS204）、関数名「`fir`」は

アセンブランプレート記憶部103に格納されているので（ステップS205）、アセンブラン関数呼出翻訳処理がなされることになる（ステップS206）。

【0045】こうして、ステップS208の判断において、未だ着目していない文がない場合、即ち、全ての文を処理し終えた場合には、プログラム変換装置100は、ソースプログラムの変換に関する動作を終了する。
<アセンブラン関数呼出翻訳処理>以下、アセンブラン関数呼出翻訳部107が行なうアセンブラン関数呼出翻訳処理について説明する。

【0046】図3は、アセンブラン関数呼出翻訳部107が行なうアセンブラン関数呼出翻訳処理の内容を示すフローチャートである。アセンブラン関数呼出翻訳部107は、メモリに格納されているソースプログラム中のアセンブランプレート関数の呼出記述を参照するために必要なメモリアドレス等の情報を制御部104から起動時に渡される。図6に示すアセンブランプレート関数の呼出記述である「`resul t1=fir(v1, v2, n)`」を参照するための情報が渡されて、アセンブラン関数呼出翻訳部107が起動される場合を例にして説明する。

【0047】アセンブラン関数呼出翻訳部107は、起動されると、ソースプログラムに記述された変数に割り当てないレジスタ、即ち退避及び復元なしで使用可能なレジスタを求め（ステップS301）、その退避及び復元なしで使用可能なレジスタを示す情報とアセンブランプレート関数の呼出記述を示す情報を渡して、アセンブランプレート選択部105を起動する。なお、変数に割り当てないレジスタがどれであるか等は従来のコンパイラにおいてなされている手法により求めることができる。

【0048】アセンブランプレート選択部105は起動されると、複数のアセンブランプレートから最適なアセンブランプレートを選択するアセンブランプレート選択処理を行い、ここでは、「`resul t1=fir(v1, v2, n)`」について「`fir`」というアセンブランプレート関数の定義として図7に示したテンプレート1を選択しアセンブラン関数呼出翻訳部107に通知する（ステップS302）。このアセンブランプレート選択処理については後に詳細に説明する。

【0049】次に、アセンブラン関数呼出翻訳部107は、アセンブランプレート選択部105により選択されたアセンブランプレート関数の定義をアセンブランコードに変換する際に引数と戻り値についての受渡しのコードが必要か否かを判断する（ステップS303）。必要である場合には、アセンブラン関数呼出翻訳部107は、引数及び戻り値について必要な受渡しのコードを生成する（ステップS304）。また、引数及び戻り値についての受渡しのコードが必要でなければステップS304をスキップする。

【0050】ステップS304で、アセンブラー関数呼出翻訳部107は、「`result1=fir(v1, v2, n)`」に対して、引数「`v1`」、「`v2`」、「`n`」についてはそれぞれ「`mov$p1, v1`」、「`mov$p2, v2`」、「`mov$p3, (n)`」という受渡しのコードを、戻り値`result1`については「`mov(result1), $t1`」という受渡しのコードを生成する。

【0051】なお、アセンブラー関数翻訳部107は、引数についての受渡しのコードは、アセンブランプレート関数の定義の本体的記述に対応して生成するアセンブラーコードの前に挿入し、戻り値についての受渡しのコードは本体的記述に対して生成するアセンブラーコードの後に挿入する。必要に応じて引数及び戻り値についての受渡しのコードを生成した後に、アセンブラー関数呼出翻訳部107は、アセンブランプレート中の割り当て候補のレジスタについての記述を参照して、アセンブランプレート関数の定義の本体的記述における仮想レジスタ「`$p3`」、「`$t1`」、「`$p1`」及び「`$p2`」に割り当てる現実のレジスタを決定し、これらの仮想レジスタを決定した現実のレジスタに置き換えることによりアセンブラーコードを生成する（ステップS305）。なお、置き換える現実のレジスタの決定は、従来のプログラム変換装置における手法と同様の手法で実現できるので、ここでは、説明を省略する。

【0052】ここでは、テンプレート1のパラメータである仮想レジスタ「`$p1`」、「`$p2`」、「`$p3`」及び「`$t1`」が、それぞれ実レジスタ「`p0`」、「`p4`」、「`r0`」及び「`a0`」に置き換えることとして説明する。次に、アセンブラー関数翻訳部107は、置き換える現実のレジスタが、既にアセンブラーコードに変換済みの部分において使用されている場合は、関数の呼出記述に関して変換する場合における従来のプログラム変換装置における手法と同様の手法により、退避及び復元のコードが必要か否かを判断する（ステップS306）。

【0053】退避及び復元が必要であると判断した場合には、アセンブラー関数翻訳部107は、レジスタの退避及び復元のアセンブラーコードを生成し、アセンブランプレート関数の定義の本体的記述に対応して生成されたアセンブラーコードの前に退避のコードを挿入し、本体的記述に対応して生成するアセンブラーコードの後に復元のコードを挿入する（ステップS307）。また、退避及び復元が必要でないと判断した場合には、ステップS307をスキップする。

【0054】ここでは、「`p0`」、「`p4`」、「`r0`」及び「`a0`」は退避及び復元の必要がなく、ステップS307の処理は行なわれないこととして説明する。最後に、アセンブラー関数翻訳部107は、アセンブランプレート関数の定義の本体的記述に対応して生成されたア

センブラーコードを出力ファイル106に出力し（ステップS308）、アセンブラー関数呼出翻訳処理を終了する。

【0055】また、図6に示すアセンブランプレート関数の呼出記述である「`result2=fir(v3, v4, 10)`」を参照するための情報が渡されて起動された場合には、アセンブラー関数翻訳部107は、基本的に上述した「`result1=fir(v1, v2, n)`」が渡された場合と同様の動作を行うが、アセンブランプレート選択部105によりなされるアセンブランプレート選択処理（ステップS302）において、図7に示したテンプレート2が選択されるため、結果的に「`result1=fir(v1, v2, n)`」に対応して生成される命令列とは異なる命令列からなるアセンブラーコードを生成する。

【0056】このようなアセンブラー関数呼出翻訳処理によって、図6に示したソースプログラムは図8に示すアセンブラーコードに変換される。プログラム変換装置100は、「`fir`」というアセンブランプレート関数を、最適なアセンブラーコードに変換するため、その呼出記述における引数に応じて、定義としてテンプレート1又はテンプレート2を選択的に採用し、図8に示すように、各呼出記述を「`repn`」と「`mac`」という命令列を中心としたアセンブラーコード又は「`macr`」という命令を中心としたアセンブラーコードに変換する。

【0057】<アセンブランプレート選択処理>以下、アセンブランプレート選択部105が行なうアセンブランプレート選択処理について説明する。図4は、アセンブランプレート選択部105が行なうアセンブランプレート選択処理の内容を示すフローチャートである。

【0058】アセンブランプレート選択部105は、アセンブラー関数翻訳部107に起動され、起動される際に、退避及び復元なしで使用可能なレジスタを示す情報とアセンブランプレート関数の呼出記述を示す情報とを渡される。ここでは、図6に示すアセンブランプレート関数の呼出記述である「`result1=fir(v1, v2, n)`」が渡される場合と、「`result2=fir(v3, v4, 10)`」が渡される場合とを例として説明する。

【0059】アセンブランプレート選択部105は、アセンブランプレート関数の呼出記述において呼出対象とされた関数名を持つ複数のアセンブランプレート関数の定義をアセンブランプレート記憶部103から獲得する（ステップS401）。アセンブランプレート選択部105は、「`result1=fir(v1, v2, n)`」や「`result2=fir(v3, v4, n)`」については図7に示した「`fir`」という関数名のテンプレート1及びテンプレート2を獲得する。

【0060】アセンブランプレート選択部105は、

獲得した複数のアセンブラーインプレート関数の定義のうち、コストが最小であるアセンブラーインプレート関数の定義を選択するものであり、このためステップS402からS407までの処理を行なう。アセンブラーインプレート選択部105は、アセンブラーインプレート関数の1つの定義に着目し（ステップS402）、アセンブラーインプレート関数の呼出記述を変換するのに使用可能な定義であるか否かを判断する（ステップS403）。

【0061】アセンブラーインプレート選択部105は、図7に示したテンプレート1に着目した場合、「`result1=fir(v1, v2, n)`」に対する場合でも「`result2=fir(v3, v4, 10)`」に対する場合でも、使用可能であると判断する。このように判断するのは、引数が定数、即ち即値でないと使用できない旨の条件、つまり後述する`mm`を用いた記述が、テンプレート1中に含まれていないからである。

【0062】しかし、アセンブラーインプレート選択部105は、図7に示したテンプレート2に着目した場合、テンプレート2の定義内の「`$p3={mm}`」は第3引数が即値でなければならないことを示しているので、第3引数が即値である呼出記述では使用可能であるが、第3引数が即値でない呼出記述では使用可能でないと判断する。従って、「`result1=fir(v1, v2, n)`」に対する場合には、「`fir`」の第3引数「`n`」が即値でないので使用可能でないと判断し、「`result2=fir(v3, v4, 10)`」に対する場合には、「`fir`」の第3引数「`10`」が即値なので使用可能であると判断する。

【0063】アセンブラーインプレート選択部105は、着目しているアセンブラーインプレート関数の定義が使用可能であると判断した場合には、その定義についてのコストを計算するコスト計算処理を行なう（ステップS404）。このコスト計算処理については後に詳細に説明する。次に、計算結果であるコストが、過去に計算したコストの中で最小のものであるか否かを判断し（ステップS405）、最小のコストであると判断した場合には、着目しているアセンブラーインプレート関数の定義を選択するときに、既にアセンブラーインプレート関数の定義が選択されているならば、それに替えて、着目しているアセンブラーインプレート関数の定義を新たに選択する。また、ステップS405において最小のコストでないと判断した場合にはステップS406をスキップして、ステップS407の判断を行う。

【0064】なお、ステップS406において、着目しているアセンブラーインプレート関数の定義を選択するときに、既にアセンブラーインプレート関数の定義が選択されているならば、それに替えて、着目しているアセンブラーインプレート関数の定義を新たに選択する。また、ステップS405において最小のコストでないと判断した場合にはステップS406をスキップして、ステップS407の判断を行う。

【0065】また、ステップS403において、着目しているアセンブラーインプレート関数の定義が使用可能で

ないと判断した場合には、ステップS404からステップS406をスキップして、ステップS407の判断を行う。ステップS407において、未だ着目していないアセンブラーインプレート関数の定義があると判断したら、次のアセンブラーインプレート関数の定義に着目すべくステップS402の処理に戻り、全てのアセンブラーインプレート関数の定義を処理し終えたならば、アセンブラーインプレート選択処理を終了する。

【0066】こうして、「`result1=fir(v1, v2, n)`」の場合は、選択候補が図7に示すテンプレート1のみであるため、アセンブラーインプレート選択部105により最終的に選択されるものはテンプレート1となる。なお、使用可能な定義が1つしかない場合においては、最小のコストのものを探すためのステップS404及びステップS405を行わない。

【0067】また、「`result2=fir(v3, v4, 10)`」の場合は、選択候補は図7に示すテンプレート1及びテンプレート2の両方であり、後述するコスト計算処理の結果として、図7に示すテンプレート1のコストが6と計算され、テンプレート2のコストが4と計算されるので、アセンブラーインプレート選択部105により最終的に選択されるものはテンプレート2となる。

【0068】<コスト計算処理>以下、コスト計算処理について説明する。図5は、アセンブラーインプレート選択部105が行うコスト計算処理の内容を示すフローチャートである。コスト計算処理として、アセンブラーインプレート選択部105は、例えば図7に示したテンプレート1では「`cost=2`」というように、アセンブラーインプレート関数の定義において記述されているコストを得てこれをコスト計算値の初期値とし（ステップS501）、引数についての受渡しのコードが必要か否かを判断する（ステップS502）。なお、この例では、「`cost`」の数値は実行サイクル数を表すものとする。

【0069】アセンブラーインプレート選択部105は、引数についての受渡しのコードが必要である場合には引数についての受渡しのコードのコストをコスト計算値に加算することにより新たにコスト計算値を求め（ステップS503）、必要でない場合にはステップS503をスキップし、次にレジスタの退避・復元のコードが必要か否かを判断する（ステップS504）。

【0070】アセンブラーインプレート選択部105は、レジスタの退避及び復元のコードが必要である場合にはレジスタの退避及び復元のコードのコストをコスト計算値に加算することにより新たにコスト計算値を求め（ステップS505）、必要でない場合には、ステップS505をスキップし、コスト計算処理を終了する。つまり、「テンプレート関数の定義の本体的記述部分のコスト」と「引数についての受渡しコードのコスト」と「レ

ジスタの退避及び復元のコードのコスト」との和が、アセンブラテンプレート関数の定義についての最終的なコスト計算値となる。

【0071】このようなコスト計算処理により、「`res1t2=fir(v3, v4, 10)`」に関する図7に示したテンプレート1のコストは、 $2+4+0=6$ と計算され、テンプレート2のコストは、 $1+3+0=4$ と計算される。引数の受け渡しのコードやレジスタの退避及び復元のコードのコストについては、一般的なプログラム変換装置と同様の処理であるところの、各種変数等をレジスタに割り付ける処理を実行した結果を踏まえて、特にアセンブラテンプレート関数の呼出記述を変換する際に新たに必要となるコードのみを計算して求める。また、これらのコードのコストは、レジスタを利用する数に応じて求めることができるように算出用の情報を予め保持していることとしてもよい。例えば、引数として変数が3つあれば、1つの戻り値の分と合わせて、コストを4とする。

＜補足＞以上、本発明に係るファイル転送システムについて、実施の形態に基づいて説明したが、本発明はこれらの実施の形態に限られないことは勿論である。即ち、(1)本実施の形態では、プログラム変換装置はアセンブラコードを出力することとしたが、アセンブラコードを機械語の形式に変換して出力することとしてもよい。(2)ソースファイル格納部101、アセンブラテンプレートファイル格納部102及び出力ファイル格納部108は、それぞれ別個のハードディスク装置等の記憶装置であっても、同一の記憶装置であってもよい。

(3)本実施の形態では、アセンブラテンプレート中に「`cost=`」という記述形式でもって、コストを明示する情報が含まれていることを表したが、このようにコストを明示しなくても、用いられる全ての命令それぞれについて、命令と、実行サイクル数、コードサイズ等のコストを示す情報を対応付けたテーブルを予めメモリ等に記憶しておく、そのテーブルを参照することによりアセンブラテンプレートに基づき変換されるであろうアセンブラコードについてのコストを算出することとしてもよい。このような方法で、アセンブラテンプレート関数の複数の定義それぞれについて、コストを算出して比較することによって、最もコストの小さい定義を選択して、その定義に従ってアセンブラテンプレート関数の呼出記述をアセンブラコードに変換することとしてもよい。

(4)本実施の形態では、コストとして実行サイクル数に着目した例に基づいてプログラム変換装置の動作について説明したが、実行サイクル数の代わりにコードサイズをコストとして用いることとしてもよい。例えば「`cost=`」の記述がコードサイズを表すように予め記述しておくこととしてもよい。

【0072】また、実行サイクル数とコードサイズとの

値を両方計算できるようにしておき、実行サイクル数とコードサイズのいずれをコストとして用いるかをプログラムの作成者その他の者による設定に従って決定しアセンブラテンプレート関数の複数の定義のうちコストを最小とする1つの定義を選択して、その定義に従ってアセンブラテンプレート関数の呼出記述をアセンブラコードに変換することとしてもよい。

(5)本実施の形態では、テンプレート2の定義内の「`$p3={imm}`」により第3引数が即値でなければならない旨の引数に関する条件が記述されていることとしたが、引数に関する条件は、引数である即値の大きさに関する条件であってもよい。例えば、「`$p3=10<imm<256`」等といった記述で、第3引数が0以上256未満の値であることを条件とする等、引数が所定範囲内の数値であること条件としてもよい。

(6)各アセンブラテンプレート関数についての複数の定義それぞれについて、その定義が選択候補となり得る条件をどのような形式で記憶していくてもよい。例えば、引数が1つしかないアセンブラテンプレート関数AについてのテンプレートA1とテンプレートA2という2つの定義がある場合に、引数が変数ならテンプレートA1で引数が即値、即ち定数ならテンプレートA2を選択候補とすることを示す選択用情報を記憶しておくこととしてもよい。この場合は、選択用情報を参照してステップS403の判断を行うことになる。また、各定義についてのコストも、どのような形式で記憶していくてもよい。

(7)本実施の形態では、退避及び復元のコードや引数についての受渡しのコードをコスト計算において考慮することとしたが、必ずしもこれらを考慮する必要はなく、テンプレート関数の定義の本体的記述部分のコストのみをコスト計算の結果とすることとしてもよい。

(8)本実施の形態におけるプログラム変換装置の処理手順(図2～図5のフローチャートの手順)等を、汎用のコンピュータ等に実行させるためのコンピュータプログラムを、記録媒体に記録し又は各種通信路等を介して、流通させ頒布することもできる。このような記録媒体には、ICカード、光ディスク、フレキシブルディスク、ROM等がある。流通、頒布されたコンピュータプログラムは、コンピュータ等にインストール等されることにより利用に供され、コンピュータは、当該コンピュータプログラムを実行して、本実施の形態で示したようなプログラム変換装置を実現する。

【0073】

【発明の効果】以上の説明から明らかなように、本発明に係るプログラム変換装置は、高級言語で記述されたソースプログラムをアセンブラコードに変換するプログラム変換装置であって、アセンブラテンプレート関数についての複数の定義を記憶するテンプレート記憶手段と、前記ソースプログラムにおけるアセンブラテンプレート関数の呼出記述を検出する呼出検出手段と、前記呼出検

出手段により前記呼出記述が検出された場合に、当該呼出記述により呼び出されるアセンブランプレート関数についての複数の定義のうち、最適な1つを選択する選択手段と、前記呼出検出手段により検出された前記呼出記述を、前記選択手段により選択されたアセンブランプレート関数の定義に基づいてアセンブラコードに変換する変換手段とを備えることを特徴とする。

【0074】これにより、アセンブランプレート関数についての複数の定義のうちの1つが選択されるので、このプログラム変換装置を利用する場合に、1つのアセンブランプレート関数に対して複数の定義を用意しておけば、ソースプログラム中のアセンブランプレート関数の呼出記述を、その記述内容等に応じた最適なアセンブラコードに変換することが可能になる。

【0075】また、前記呼出検出手段は、前記ソースプログラム中の関数の呼出記述を検出して、当該呼出記述により呼び出される関数についての定義が前記テンプレート記憶手段に記憶されているか否かを判断し、定義が記憶されている場合に当該呼出記述をアセンブランプレート関数の呼出記述として検出することとしてもよい。

【0076】これにより、ソースプログラム上において通常の関数の呼出記述と同様の形式でもって記述されたアセンブランプレート関数の呼出記述を検出して、最適なアセンブラコードに変換することができるようになる。また、前記テンプレート記憶手段は、前記アセンブランプレート関数の定義毎に、関数への引数についての条件がある場合にはその条件を示す条件情報をも記憶しており、前記選択手段は、前記条件情報に従い、前記呼出記述に含まれる関数への引数の記述に応じて前記選択を行うこととしてもよい。

【0077】これにより、アセンブランプレート関数の呼出記述において、変数を引数としているか、即値、即ち定数を引数としているか等、引数の別に対応して、アセンブランプレート関数の呼出記述をより最適なアセンブラコードに変換することができるようになる。また、前記条件情報は、引数が即値でなければならない旨を示す情報であり、前記選択手段は、前記条件情報に従い、前記呼出記述に含まれる関数への引数の記述が即値であるか否かに応じて前記選択を行うこととしてもよい。

【0078】これにより、アセンブランプレート関数の呼出記述において、変数を引数としているか、定数を引数としているかに応じて異なるアセンブラコードを生成すること、即ちアセンブランプレート関数の呼出記述をより最適なアセンブラコードに変換することができるようになる。また、前記条件情報は、引数が所定範囲値内の即値でなければならない旨を示す情報であり、前記選択手段は、前記条件情報に従い、前記呼出記述に含まれる関数への引数の記述が所定範囲内の即値であるか

否かに応じて前記選択を行うこととしてもよい。

【0079】これにより、アセンブランプレート関数の呼出記述において、定数を引数としていても、その定数が所定の範囲の値であるか否かによって、生成するアセンブラコードを異なるものとすることができますようになる。例えば、0から255まで等の1バイトで表せる数値範囲の定数であれば、1バイトの即値が指定できる命令が使えるが、256以上の数値の定数であれば、2バイトの数値を指定する命令を使わなければならないといったような場合において、0から255までの数値の定数を引数としたアセンブランプレート関数の呼出記述を、2バイトの数値を指定する命令を含むアセンブラコードに変換するような無駄を防止することができるようになる。

【0080】また、前記テンプレート記憶手段は、前記アセンブランプレート関数の定義毎に、最適なものを選択するための基準となる評価値をも記憶しており、前記選択手段は、前記各定義についての評価値に基づいて、前記選択を行うこととしてもよい。これにより、アセンブランプレート関数についての複数の定義のうち最適な定義を高速に選択することが可能となり、プログラムの変換が迅速に行えるようになる。

【0081】また、前記テンプレート記憶手段は、前記定義毎に当該定義が特定するアセンブラコードについての実行サイクル数を示す情報を前記評価値として記憶しており、前記選択手段は、前記複数の定義のうち、実行サイクル数が最小であるアセンブラコードに変換されることになる1つの定義を選択することとしてもできる。

【0082】これにより、アセンブランプレート関数の呼出記述を実行速度の速いアセンブラコードに変換することができるようになる。また、前記テンプレート記憶手段は、前記定義毎に当該定義が特定するアセンブラコードについてのコードサイズを示す情報を前記評価値として記憶しており、前記選択手段は、前記複数の定義のうち、コードサイズが最小であるアセンブラコードに変換されることになる1つの定義を選択することとしてもよい。

【0083】これにより、アセンブランプレート関数の呼出記述をサイズの小さなアセンブラコードに変換することができるようになる。また、前記選択手段は、前記呼出記述を仮に前記定義それぞれにより特定されるアセンブラコードに変換するとした場合に、引数をレジスタに設定するためのコード並びにレジスタ退避及び復元のためのコードのうち必要となるコードがあればそのコードをも含めて得られる変換結果のコードについての実行サイクル数又はコードサイズに基づいて判断することにより、前記選択を行うこととしてもよい。

【0084】これにより、コスト計算を厳密に行うことが可能となり、コストの最小となる最適な定義を選択す

ることができ、この結果として、アセンブラテンプレート関数の呼出記述を最適なアセンブラコードに変換することができるようになる。

【図面の簡単な説明】

【図1】本発明の実施の形態に係るプログラム変換装置100の機能ブロック図である。

【図2】プログラム変換装置100の動作を示すフローチャートである。

【図3】アセンブラ関数呼出翻訳部107が行なうアセンブラ関数呼出翻訳処理の内容を示すフローチャートである。

【図4】アセンブラテンプレート選択部105が行なうアセンブラテンプレート選択処理の内容を示すフローチャートである。

【図5】アセンブラテンプレート選択部105が行なうコスト計算処理の内容を示すフローチャートである。

【図6】アセンブラテンプレート関数を呼び出す旨の記述を含むソースプログラムの例を示す図である。

【図7】アセンブラテンプレートの内容例を示す図である。

【図8】本発明に係るプログラム変換装置100により図6に示すソースプログラムを変換した結果のアセンブラコードを示す図である。

【図9】従来のプログラム変換装置900の機能ブロック図である。

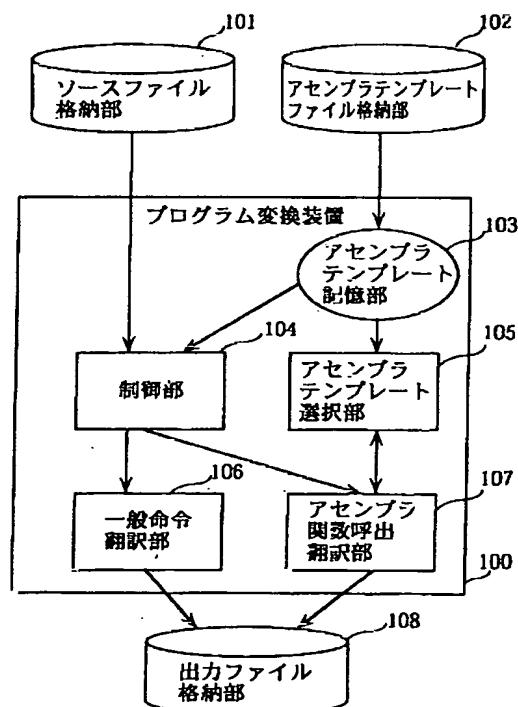
【図10】従来のアセンブラテンプレートの内容例を示す図である。

【図11】従来のプログラム変換装置900により図6に示すソースプログラムを変換した結果のアセンブラコードを示す図である。

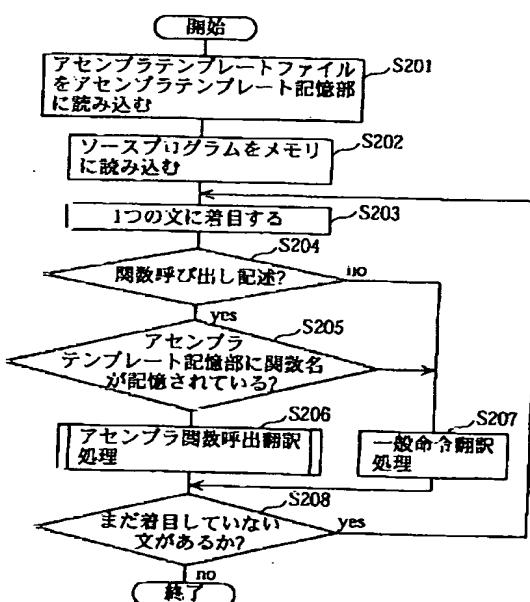
【符号の説明】

- 100 プログラム変換装置
- 101 ソースファイル格納部
- 102 アセンブラテンプレートファイル格納部
- 103 アセンブラテンプレート記憶部
- 104 制御部
- 105 アセンブラテンプレート選択部
- 106 一般命令翻訳部
- 107 アセンブラ関数呼出翻訳部
- 108 出力ファイル格納部

【図1】



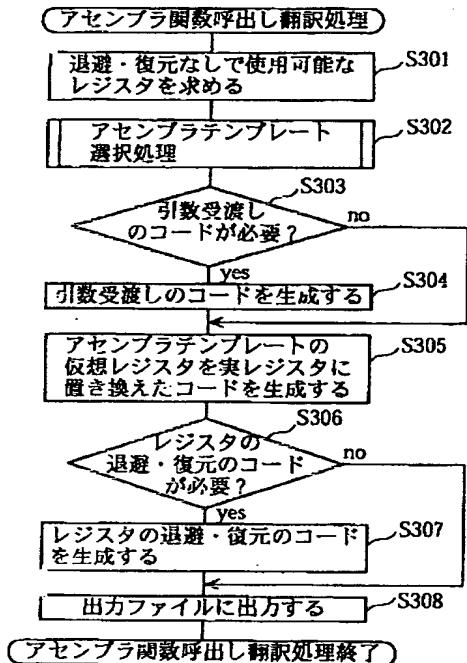
【図2】



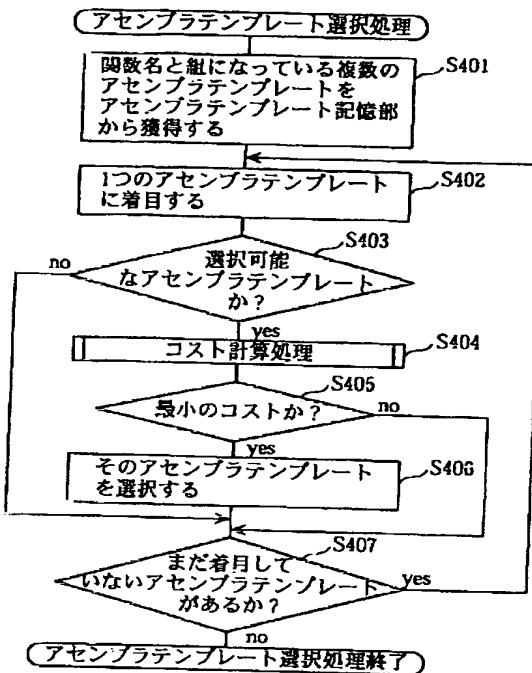
【図6】

```
void f(void)
{
    result1 = fir(v1, v2, n);
    result2 = fir(v3, v4, 10);
}
```

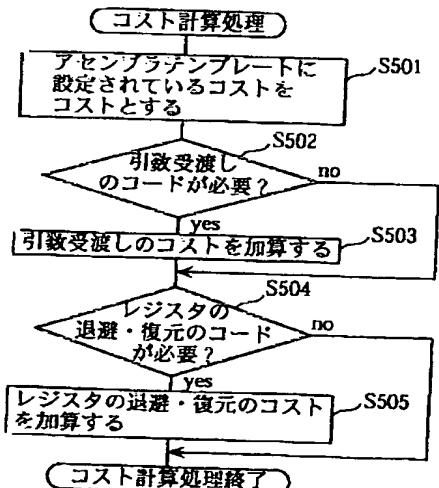
【図3】



【図4】



【図5】



【図7】

```

// テンプレート1
asm_template fir (
    $p1 = ( p0, p1, p2, p3 )
    $p2 = ( p4, p5, p6, p7 )
    $p3 = ( r0, r1, r2, r3 )
    $t1 = ( a0, a1 )
    ret = ( $t1 )
    kill = ( $p1, $p2 )
    code = (
        repn    $p3
        mac    $t1, m($p1, 1), m($p2, 1)
    )
    cost = 2
)

// テンプレート2
asm_template fir (
    $p1 = ( p0, p1, p2, p3 )
    $p2 = ( p4, p5, p6, p7 )
    $p3 = ( imm )
    $t1 = ( a0, a1 )
    ret = ( $t1 )
    kill = ( $p1, $p2 )
    code = (
        macr    $p3, $t1, m($p1, 1), m($p2, 1)
    )
    cost = 1
)
  
```

【図8】

```

MOV    p0, v1
MOV    p4, v2
MOV    r0, _n
REPNE  r0
MAC    a0, m(p0, 1), m(p4, 1)
MOV    (_result1), a0

MOV    p0, v3
MOV    p4, v4
MACR   10, a0, m(p0, 1), m(p4, 1)
MOV    (_result2), a0

```

【図10】

```

asm template fir {
    $p1 = ( p0, p1, p2, p3 )
    $p2 = ( p4, p5, p6, p7 )
    $p3 = ( r0, r1, r2, r3 )
    $t1 = ( a0, a1 )
    ret = ( $t1 )
    kill = ( $p1, $p2 )
    code = {
        REPNE  $p3
        MAC    $t1, m($p1, 1), m($p2, 1)
    }
}

```

【図11】

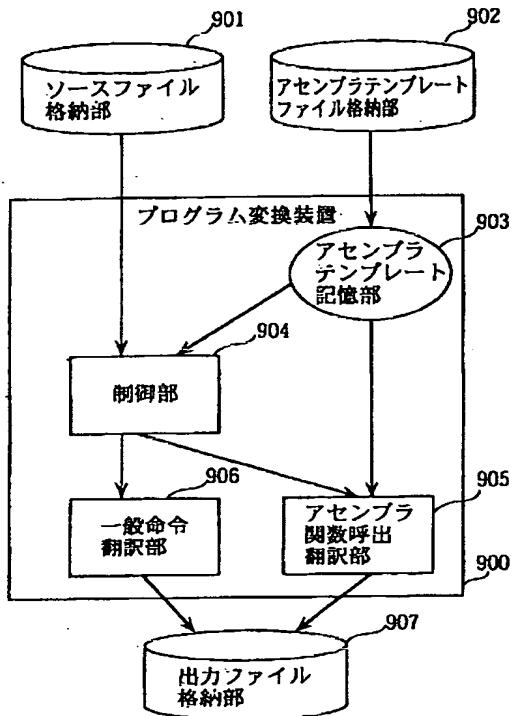
```

MOV    p0, v1
MOV    p4, v2
MOV    r0, _n
REPNE  r0
MAC    a0, m(p0, 1), m(p4, 1)
MOV    (_result1), a0

MOV    p0, v3
MOV    p4, v4
MOV    r0, 10
REPNE  r0
MAC    a0, m(p0, 1), m(p4, 1)
MOV    (_result2), a0

```

【図9】



フロントページの続き

(72)発明者 橋口 渉

大阪府門真市大字門真1006番地 松下電器
産業株式会社内

Fターム(参考) 5B081 CC21